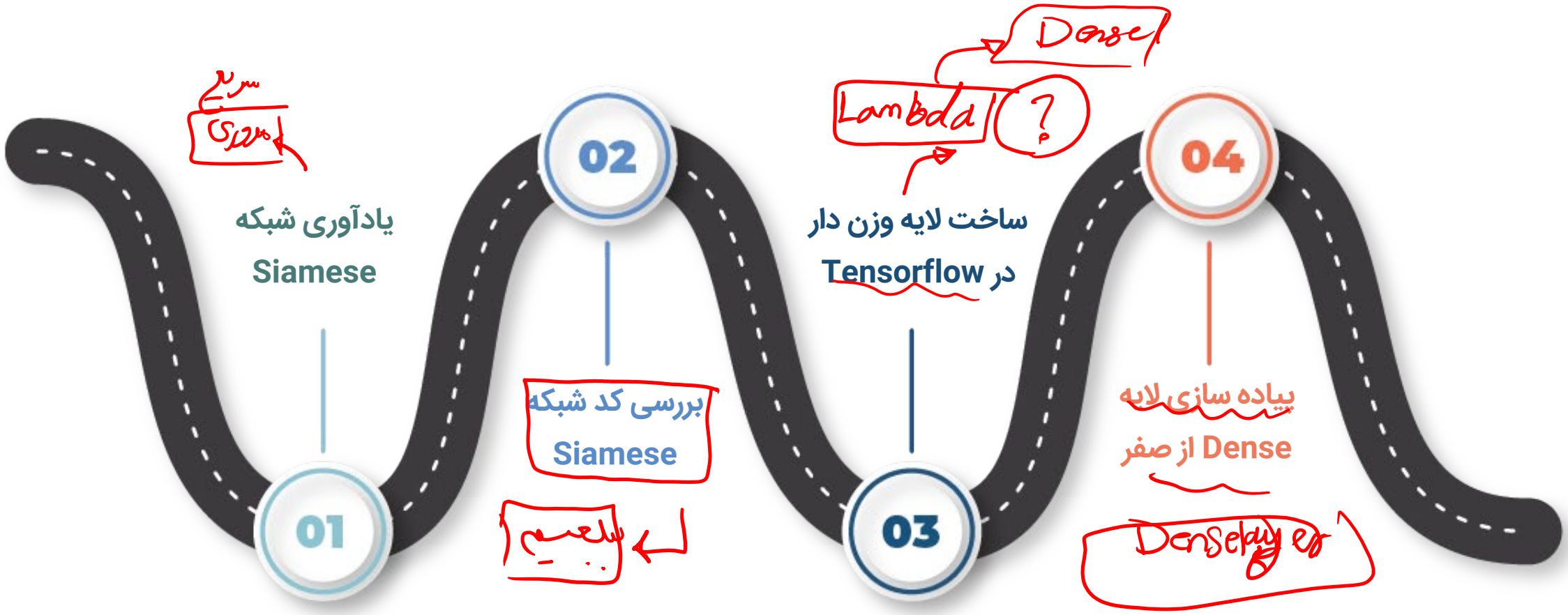
آکادمی ربوتک

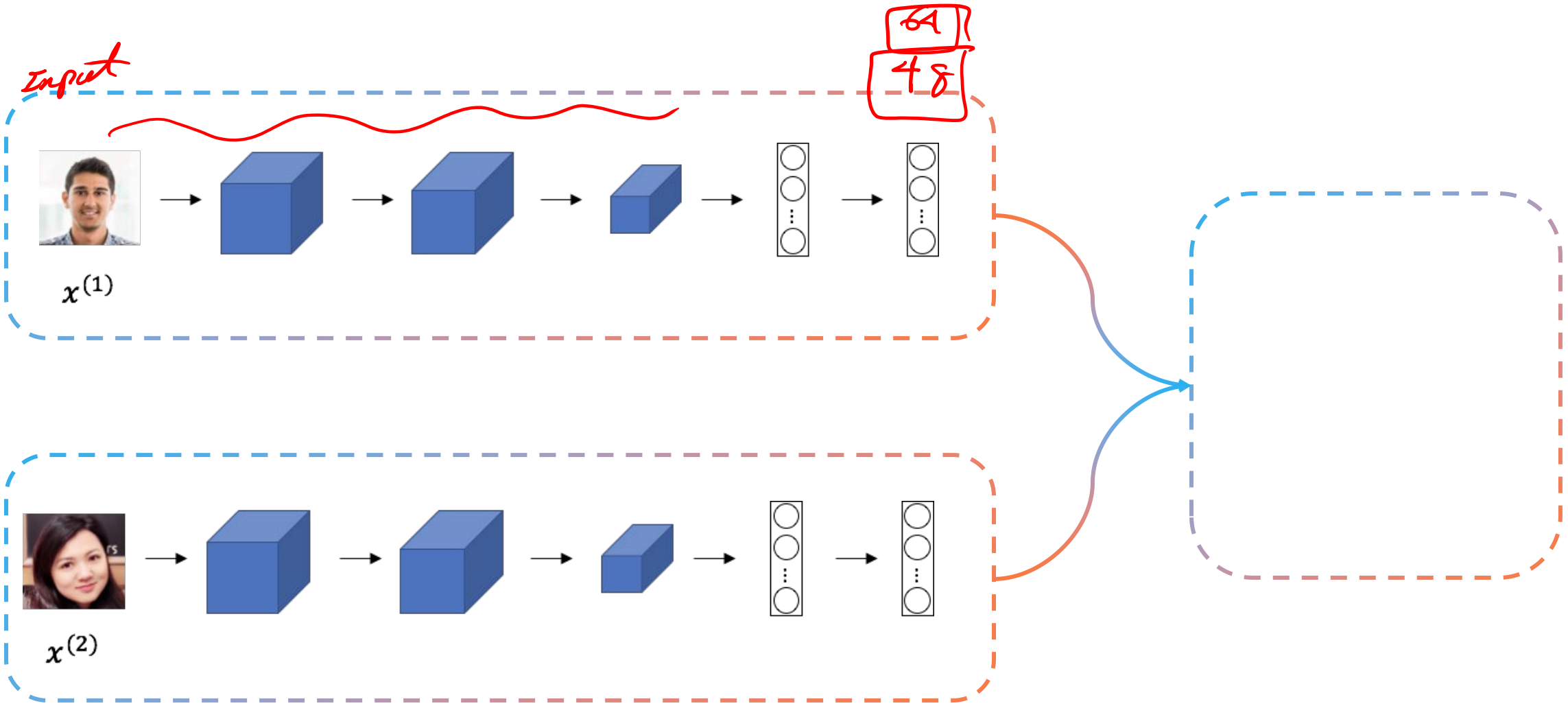
دوره تنسورفلو پیشرفته

جلسه چهارم : Network Siamese و ساخت لایه وزن دار

آنچه امروز خواهیم گفت :



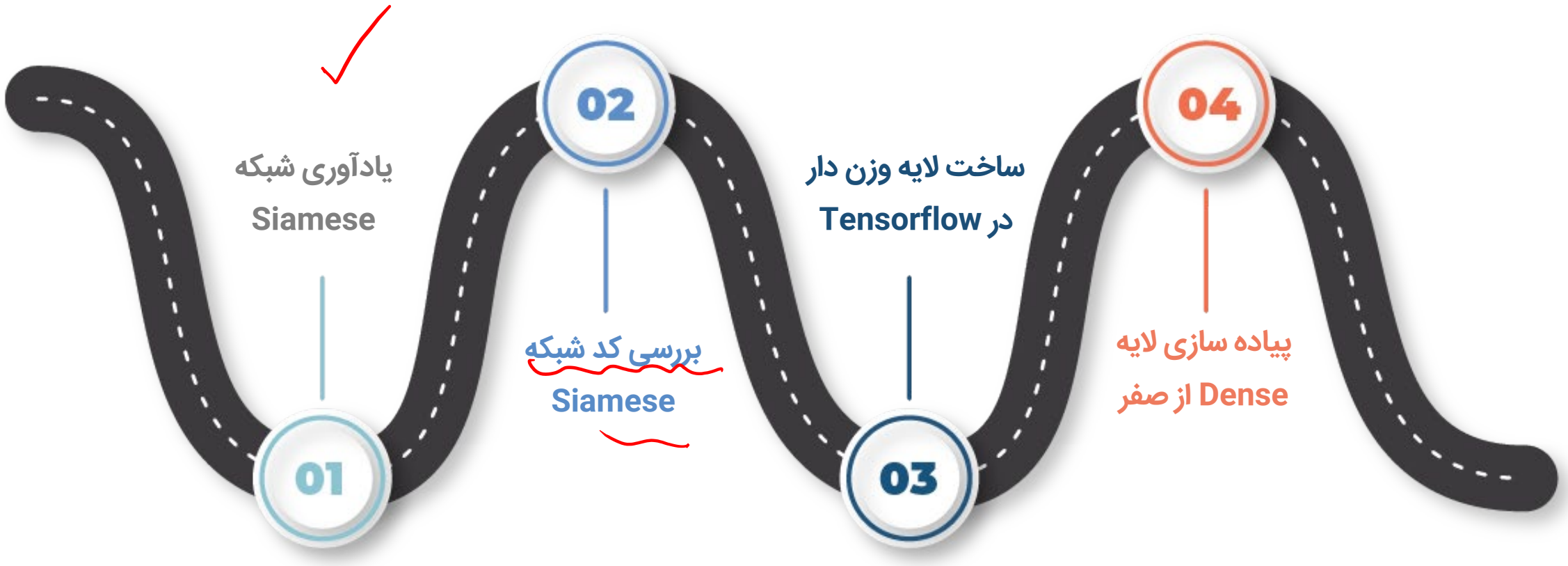
یادآوری شبکه Siamese



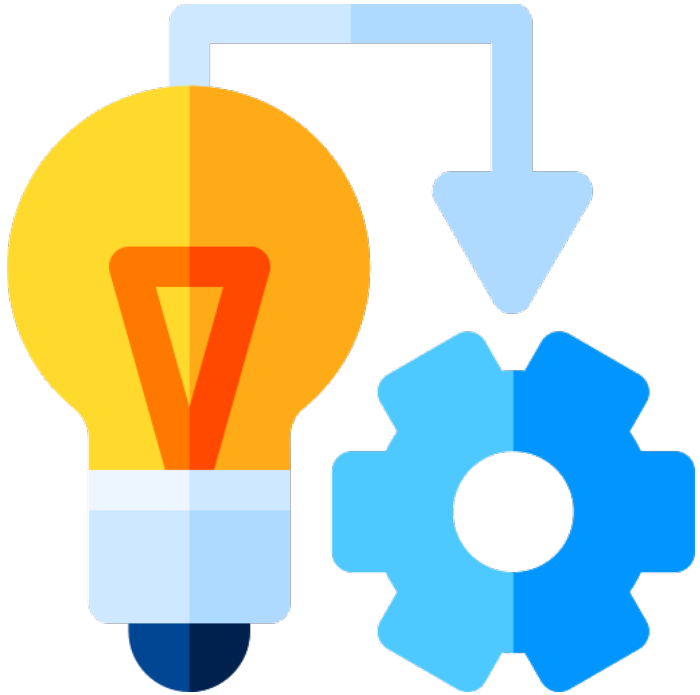
Contrastive Loss

$$Loss = y\|f(A) - f(B)\|^2 + (1 - y)\max(0, m - \|f(A) - f(B)\|^2)$$

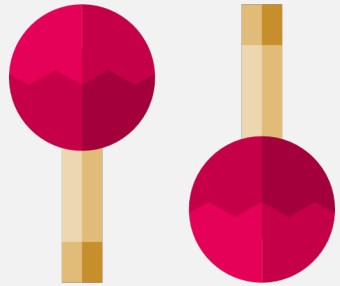
آنچه امروز خواهیم گفت :



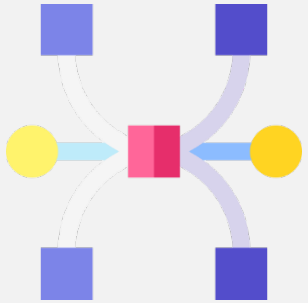
حالا برویم سر وقت پیاده سازی Siamese Network ها



برای پیاده سازی شبکه Siamese باید چند کار انجام دهیم:



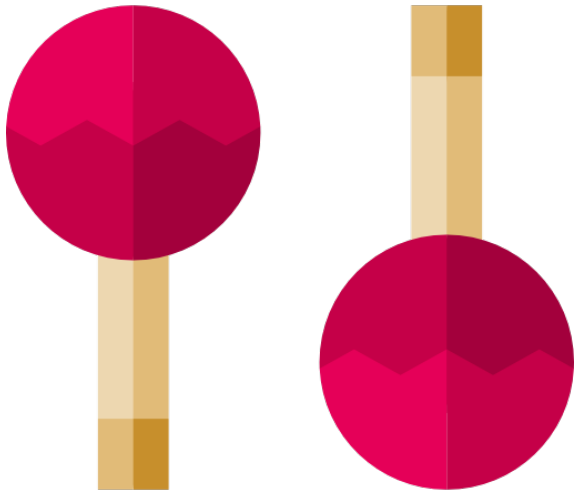
ساخت Image Pair ها



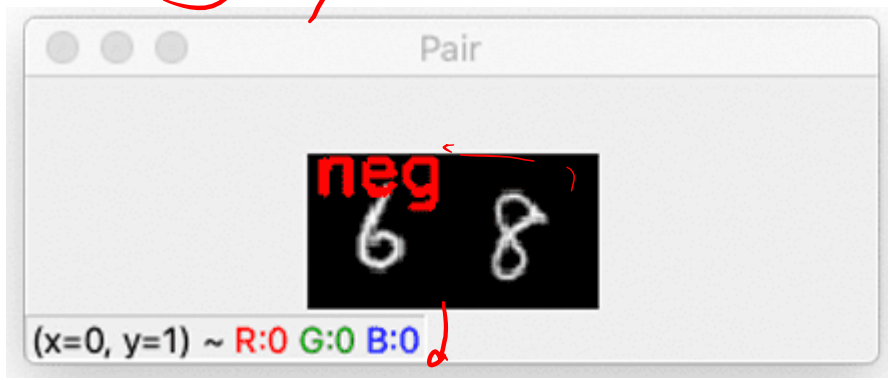
نوشتن لایه اقلیدسی

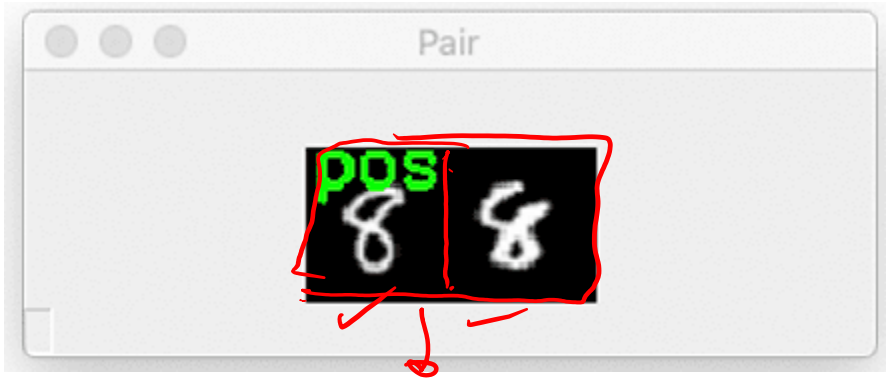
نوشتن Loss Function

ساخت Image Pair ها



خروجی این مرحله چیست؟



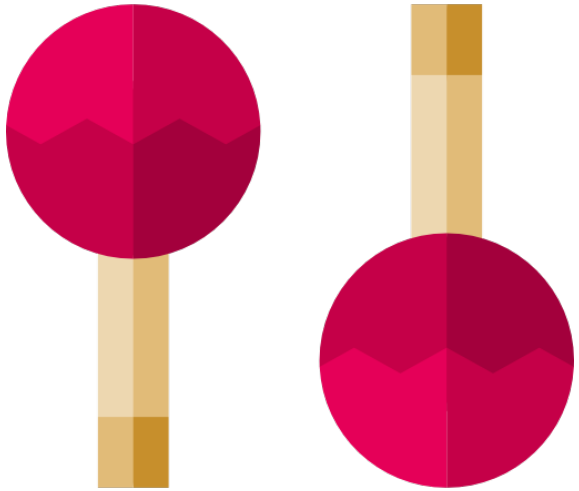


اضافه کردن کتابخانه های مورد نیاز

```
from tensorflow.keras.datasets import mnist
from imutils import build_montages
import numpy as np
import cv2
```

همون مثل

تابع Image Pair



```
def make_pairs(images, labels):


    pairImages = []
    pairLabels = []
```

پیدا کردن تعداد کلاس ها و کمی کار پیچیده !

```
numClasses = len(np.unique(labels))
idx = [np.where(labels==i)[0], for i in range(0, numClasses)]
```

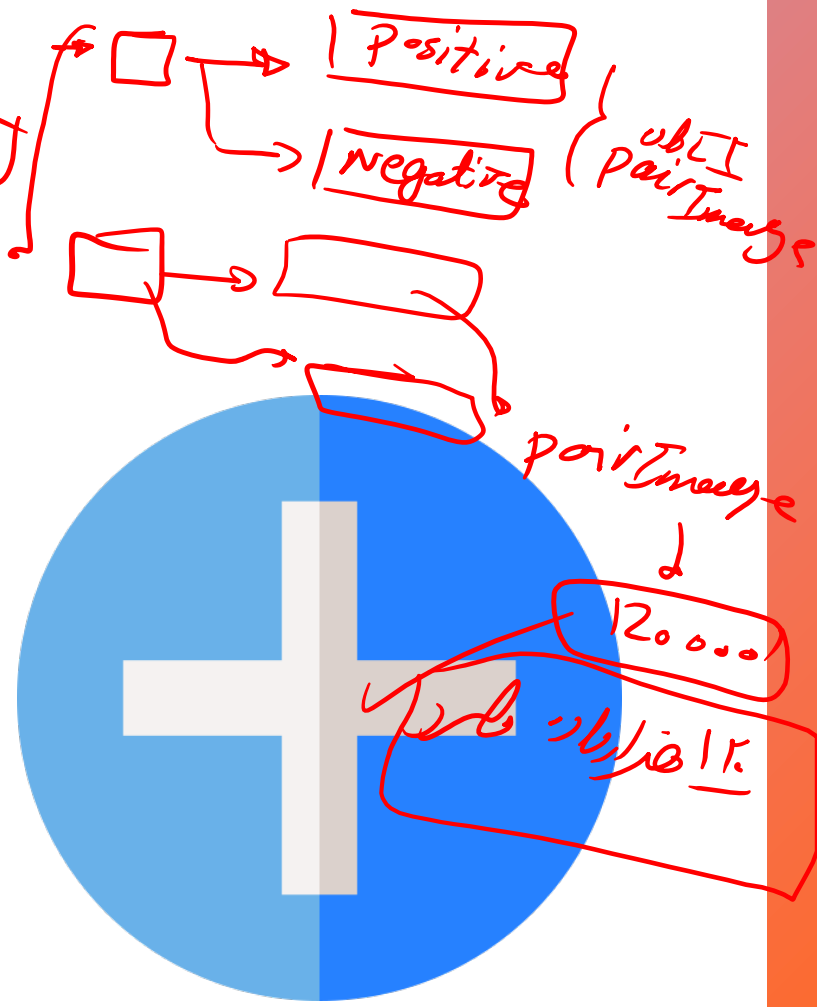
چه اتفاقی دارد می افتد ؟

```
for i in range(0, 10):
    idxs = np.where(y_test == i)[0]
    print("{}:{} {}".format(i, len(idxs), idxs))
```

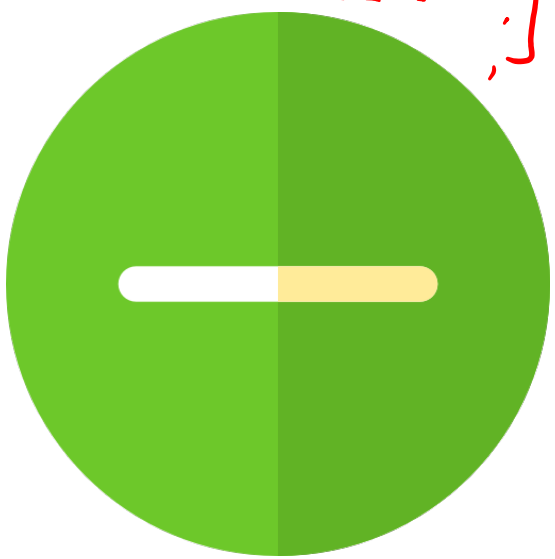
0: 5923 [1 21 34 ... 59952 59972 59987]
1: 6742 [3 6 8 ... 59979 59984 59994]
2: 5958 [5 16 25 ... 59983 59985 59991]
3: 6131 [7 10 12 ... 59978 59980 59996]
4: 5842 [2 9 20 ... 59943 59951 59975]
5: 5421 [0 11 35 ... 59968 59993 59997]
6: 5918 [13 18 32 ... 59982 59986 59998]
7: 6265 [15 29 38 ... 59963 59977 59988]
8: 5851 [17 31 41 ... 59989 59995 59999]
9: 5949 [4 19 22 ... 59973 59990 59992]

ساخت pair های مثبت

```
for idxA in range(len(images)):

    currentImage = images[idxA]
    label = labels[idxA]
    idxB = np.random.choice(idx[label])
    posImage = images[idxB]

    pairImages.append([currentImage, posImage])
    pairLabels.append([1])
```



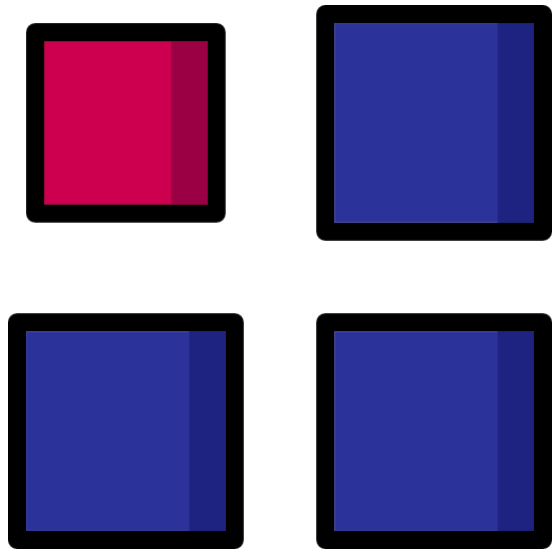
ساخت pair های منفی



```
negIdx = np.where(labels != label)[0]
negImage = images[np.random.choice(negIdx)]

pairImages.append([currentImage, negImage])
pairLabels.append([0])
```

تبدیل به آرایه کنیم:

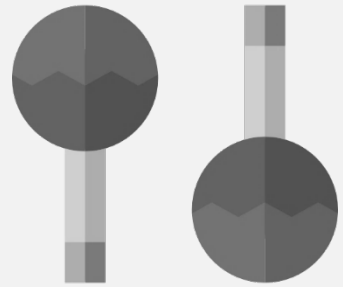
```
return (np.array(pairImages), np.array(pairLabels))
```



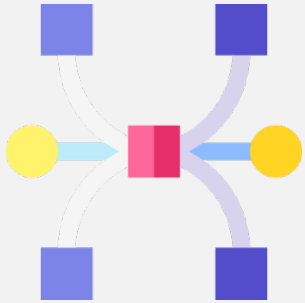
فراخوانی و استفاده از تابع:

```
(pairTrain, labelTrain) = make_pairs(X_train, y_train)
(pairTest, labelTrain)  = make_pairs(X_test, y_test)
```

برای پیاده سازی شبکه Siamese باید چند کار انجام دهیم:



ساخت Image Pair ها

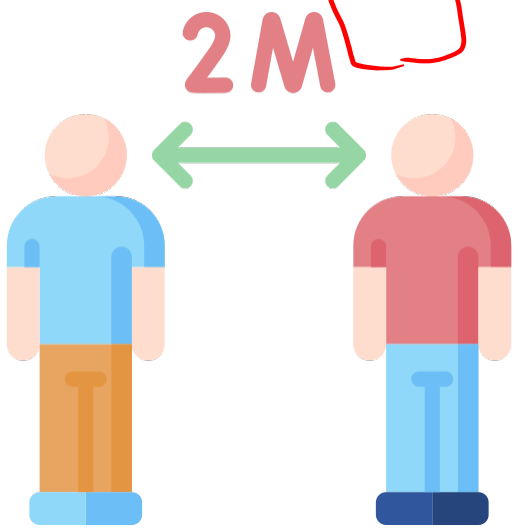


نوشتن لایه اقلیدسی

نوشتن Loss Function

تابع محاسبه فاصله اقلیدسی:

```
def euclidean_distance(vectors):

    (featsA, featsB) = vectors

    return tf.math.reduce_euclidean_norm(featsA - featsB, axis = 1, keepdims = True)
```



برای پیاده سازی شبکه Siamese باید چند کار انجام دهیم:

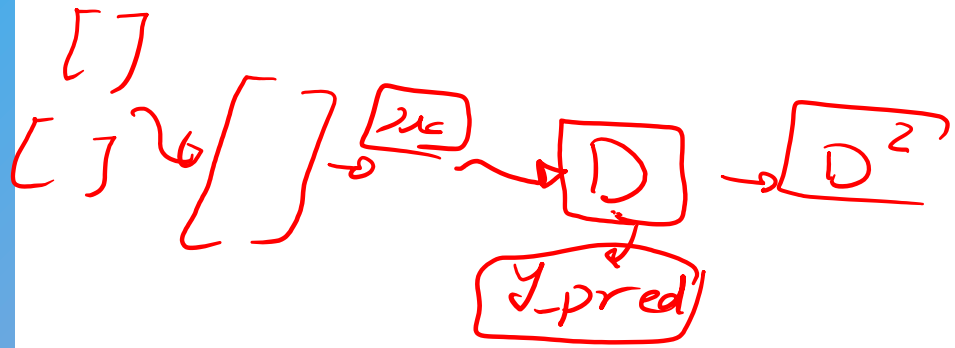
Contrastive Loss یادآوری

$$Loss = y\|f(A) - f(B)\|^2 + (1-y)\max(0, m - \|f(A) - f(B)\|^2)$$

$$Loss = yD^2 + (1-y)\max(0, m - D^2)$$

کمی دقیق تر برای پیاده سازی

$$Loss = yD^2 + (1-y)\max(0, m - D^2)$$



$$Loss = y_{true}\, y_{pred}^2 + (1 - y_{true})\max(0, m - y_{pred}^2)$$

پیاده سازی ساده Contrastive Loss

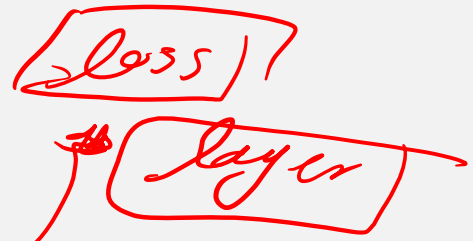
```
def contrastive_loss(y_true, y_pred):


    margin = 1
    square_pred = tf.square(y_pred)
    margin_square = tf.maximum(margin-square_pred ), 0)
    return tf.reduce_mean(y_true * square_pred + (1-y_true)*margin_square)
```

پیاده سازی با پارامتر

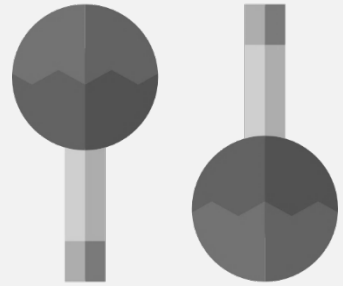
```
def contrastive_loss_with_margin(margin):

    def contrastive_loss(y_true, y_pred):
        square_pred = tf.square(y_pred)
        margin_square = tf.square(tf.maximum(margin-square_pred ), 0)
        return tf.reduce_mean(y_true * square_pred + (1-y_true)*margin_square)


    return contrastive_loss
```

پیاده سازی با OOP

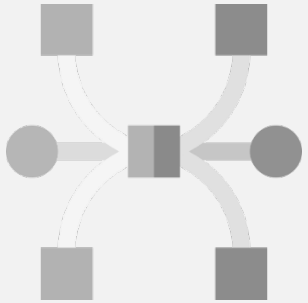
```python
class ContrastiveLoss(Loss):
    def __init__(self, margin):
        super().__init__()
        self.margin = margin

    def call(self, y_true, y_pred):
        square_pred = tf.square(y_pred)
        margin_square = tf.square(tf.maximum(self.margin-y_pred), 0)
        return tf.reduce_mean(y_true * square_pred + (1-y_true)*margin_square)
```



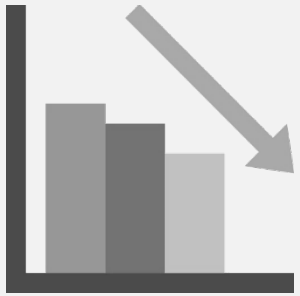
برای پیاده سازی شبکه Siamese باید چند کار انجام دهیم:



ساخت Image Pair ها

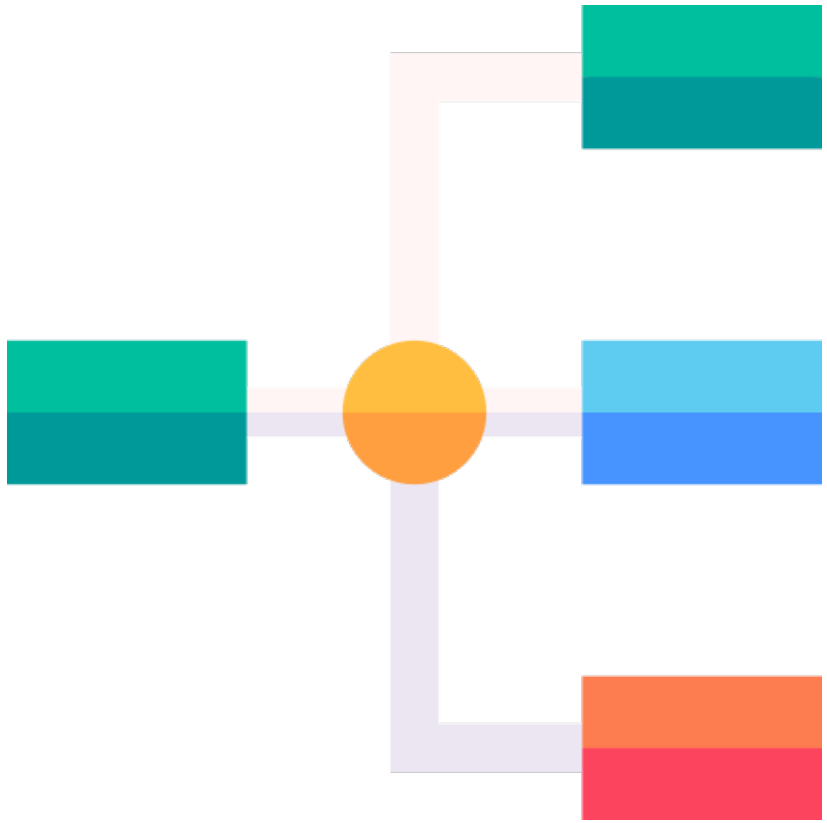


نوشتن لایه اقلیدسی

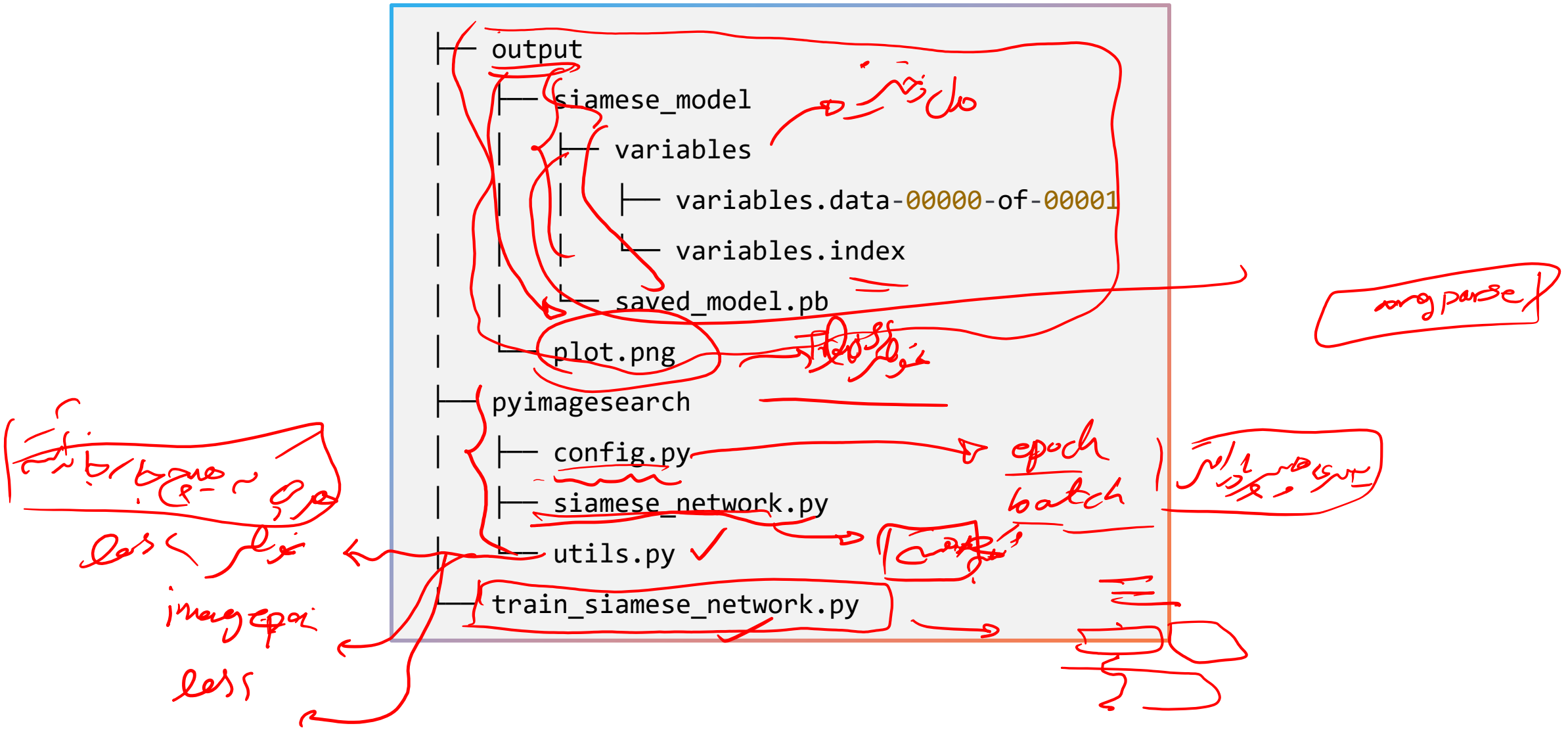


نوشتن Loss Function

بررسی ساختار کلی پروژه



ساختار کلی :


output
siamese_model
variables
variables.data-00000-of-00001
variables.index
saved_model.pb
plot.png
pyimagesearch
config.py
siamese_network.py
utils.py
train_siamese_network.py

شروع کنیم با یک کار زیبا: (فایل config)

```
IMG_SHAPE = (28, 28, 1)
BATCH_SIZE = 64
EPOCHS = 100
BASE_OUTPUT = "output"
MODEL_PATH = os.path.sep.join([BASE_OUTPUT, "siamese_model"])
PLOT_PATH = os.path.sep.join([BASE_OUTPUT, "plot.png"])
```

بریم سراغ فایل siamese_network.py

```
def build_siamese_model(inputShape, embeddingDim=48):
    inputs = Input(inputShape)
    x = Conv2D(64, (2, 2), padding="same", activation="relu")(inputs)
    …
    x = Dropout(0.3)(x)
    pooledOutput = GlobalAveragePooling2D()(x)
    outputs = Dense(embeddingDim)(pooledOutput)
    model = Model(inputs, outputs)
    return model
```

حالا فایل utils.py

```
def make_pairs(images, labels):

def euclidean_distance(vectors):

def plot_training(H, plotPath):
```

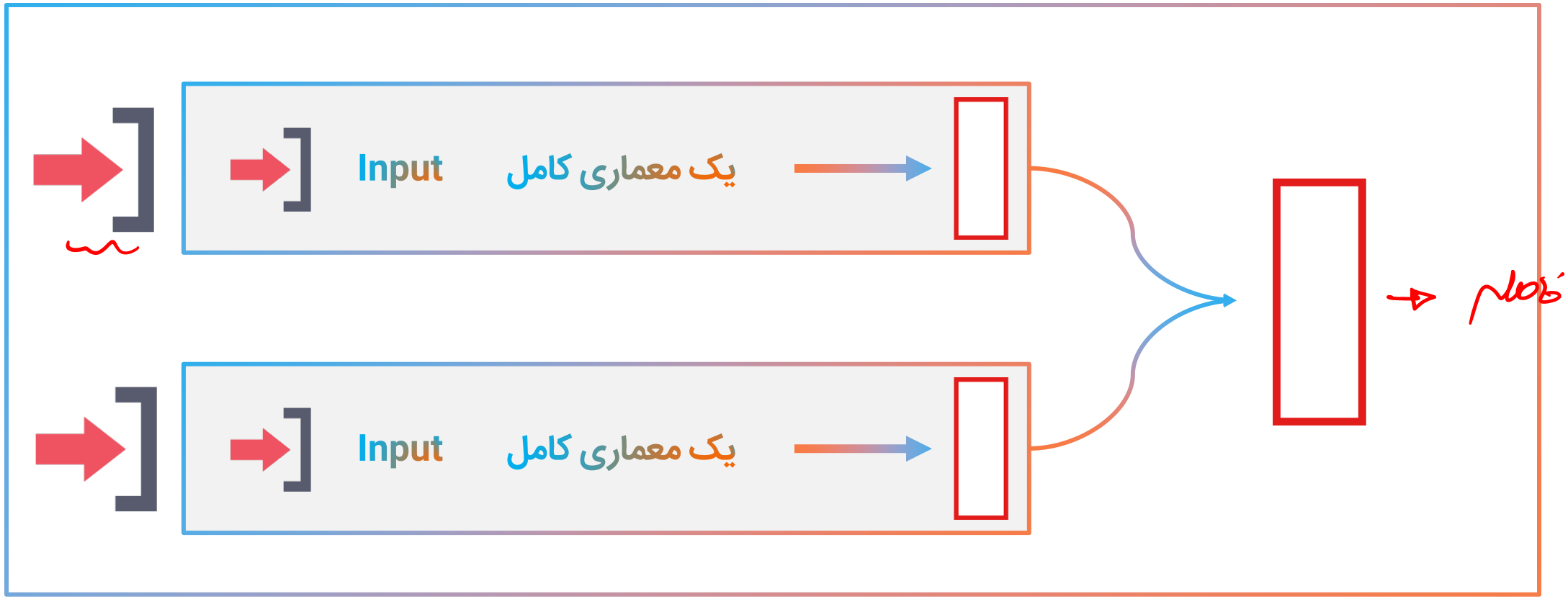
بنابراین تا اینجای کار:

```
├── output
│   ├── siamese_model
│   │   ├── variables
│   │   │   ├── variables.data-00000-of-00001
│   │   │   └── variables.index
│   │   └── saved_model.pb
│   └── plot.png
├── pyimagesearch
│   ├── config.py
│   ├── siamese_network.py
│   └── utils.py
└── train_siamese_network.py
```

تعریف ساختار ماژولار معماری Siamese

برای تعریف Siamese ابتدا شبکه های Sister را تعریف می کنند:

ساختار کامل شبکه Siamese



در پرانتز:

```
o = layers.Lambda(custom_function)(x)
```

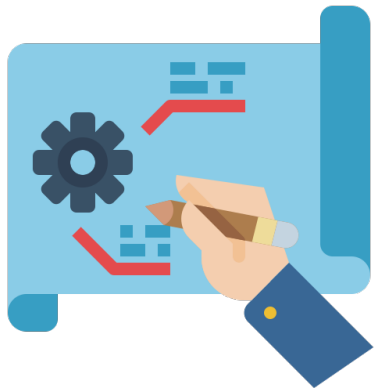
برای یک ورودی

```
o = layers.Lambda(custom_function)([x, y])
```

برای چند ورودی

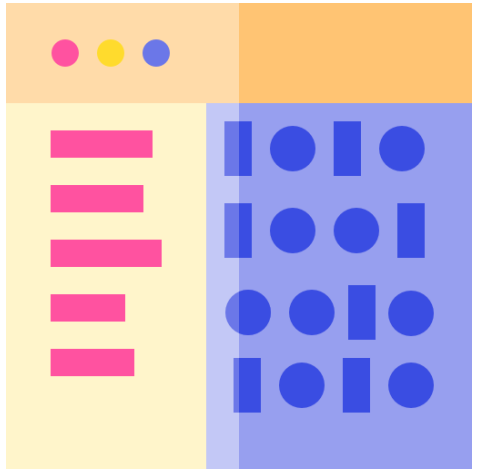
معماری کلی شبکه siamese

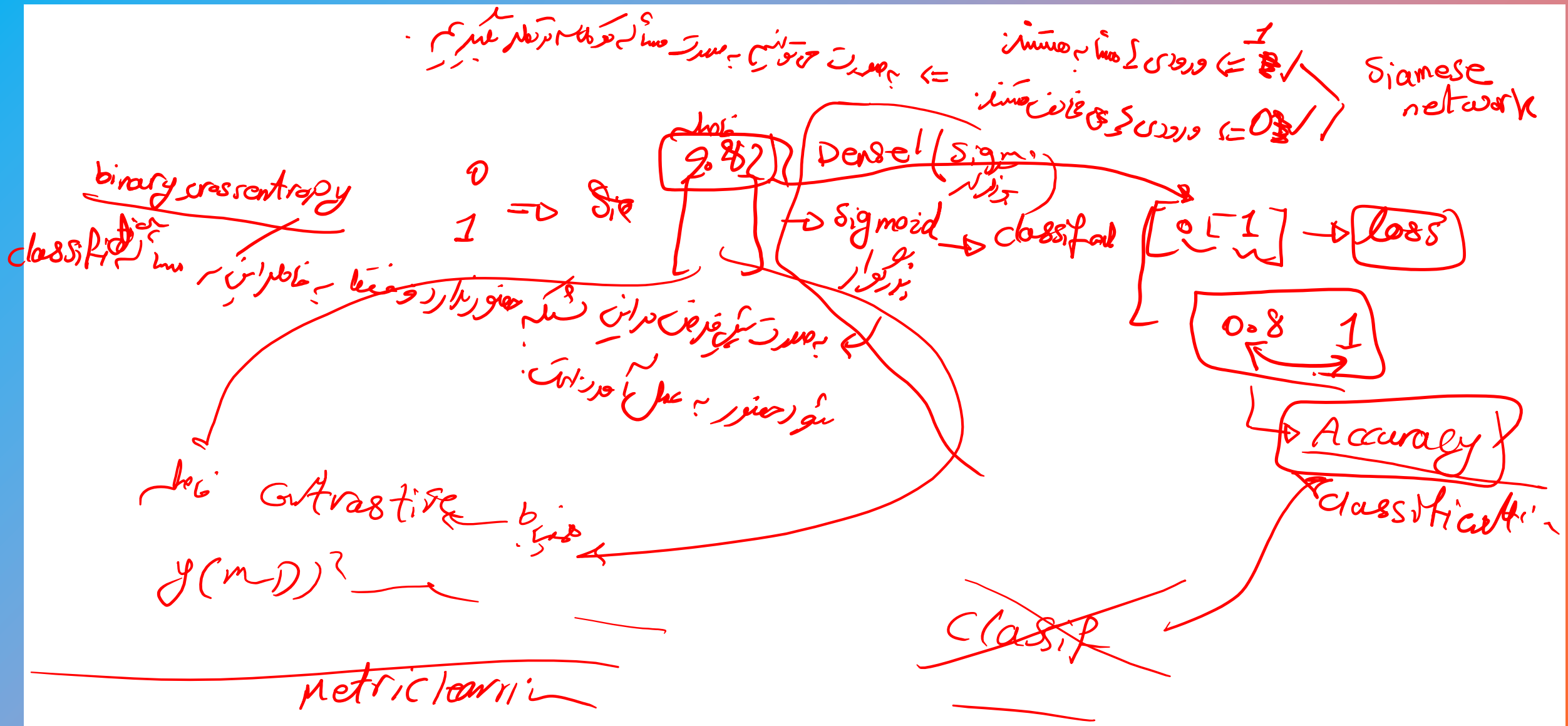
```
imgA = Input(shape=config.IMG_SHAPE)
imgB = Input(shape=config.IMG_SHAPE)
featureExtractor = build_siamese_model(config.IMG_SHAPE)
featsA = featureExtractor(imgA)
featsB = featureExtractor(imgB)
distance = Lambda(utils.euclidean_distance)([featsA, featsB])
outputs = Dense(1, activation="sigmoid")(distance)
model = Model(inputs=[imgA, imgB], outputs=outputs)
```



کامپایل و آموزش مدل

```
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
print("[INFO] Training the model...")
history = model.fit([pairTrain[:, 0], pairTrain[:, 1]], labelTrain[:],
                    validation_data=([pairTest[:, 0], pairTest[:, 1]], labelTest[:]),
                    batch_size=config.BATCH_SIZE, epochs=config.EPOCHS)
```



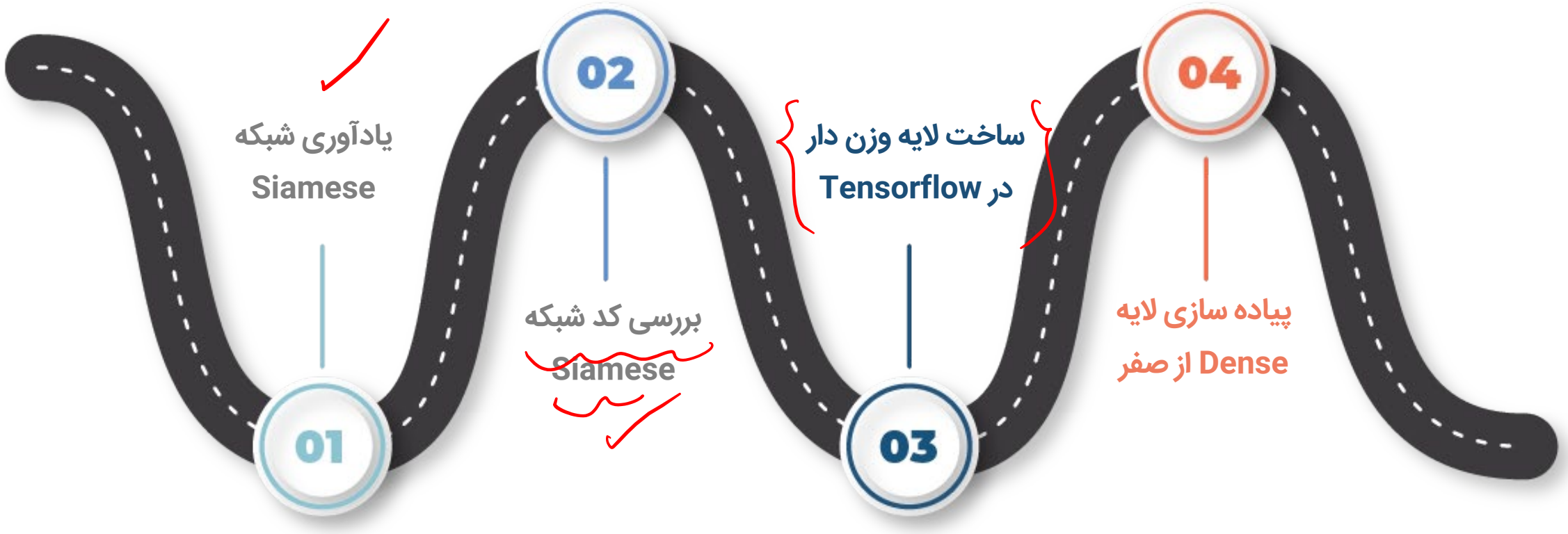

Siamese network
Dense(1, sigmoid)
2048
sigmoid
binary crossentropy
0
1
Accuracy
loss
0.8
1
Metric learning

چند درصد تا اینجا ؟

```
├── output
│   ├── siamese_model
│   │   ├── variables
│   │   │   ├── variables.data-00000-of-00001
│   │   │   └── variables.index
│   │   └── saved_model.pb
│   └── plot.png
├── pyimagesearch
│   ├── config.py
│   ├── siamese_network.py
│   └── utils.py
└── train_siamese_network.py
```

استفاده از Contrastive Loss

آنچه امروز خواهیم گفت :



نوشتن لایه اختصاصی در Tensorflow

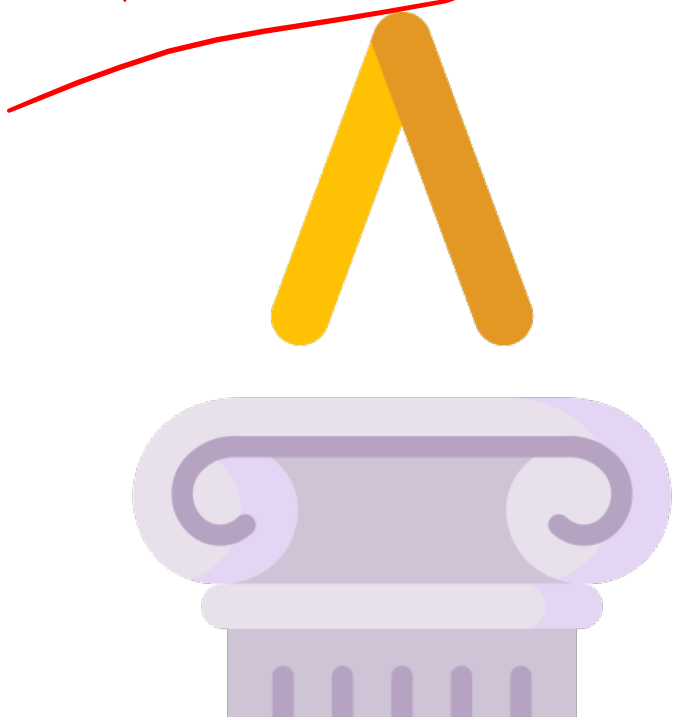
انواع روش های نوشتن لایه در Tensorflow

Trainable Layer

Lambda Layer

مشکل اصلی ما با Lambda Layer چیست؟

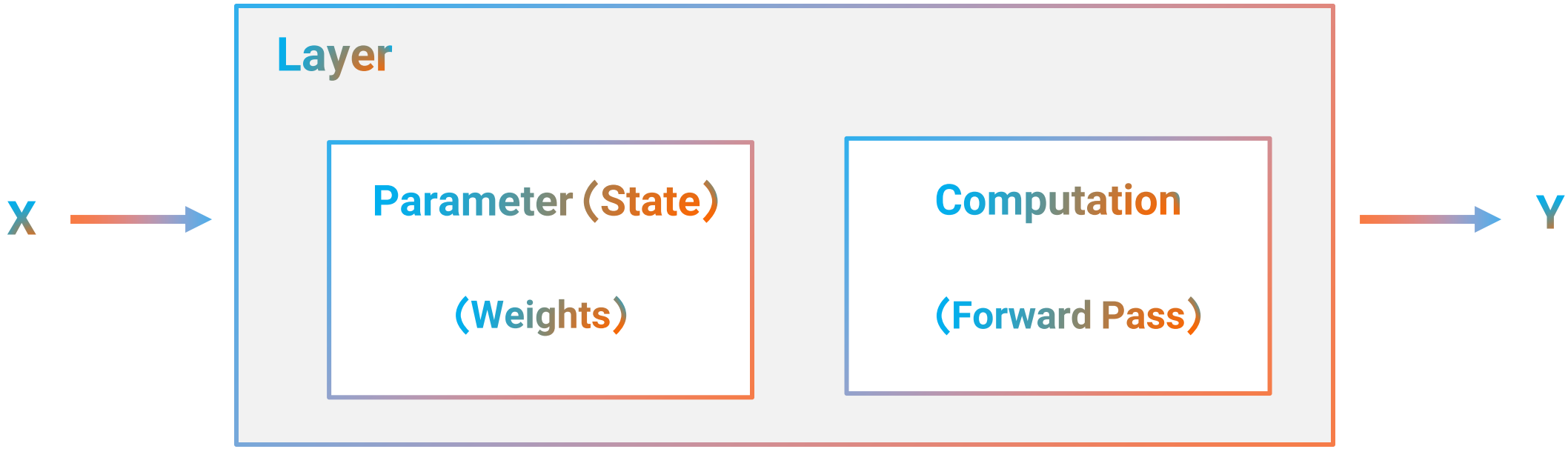
Trainable



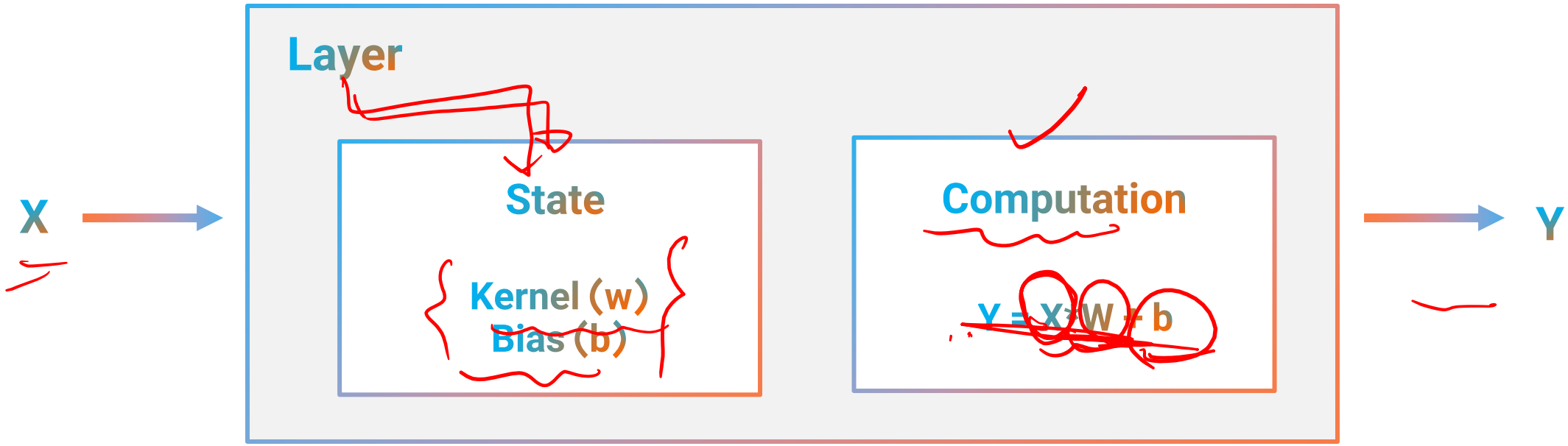
یک لایه در Keras

یک لایه در واقع یک کلاس است که تعدادی پارامتر دارد و با انجام محاسباتی بر روی ورودی خود، یک خروجی را تولید می کند.

با زبانی شفاف تر:



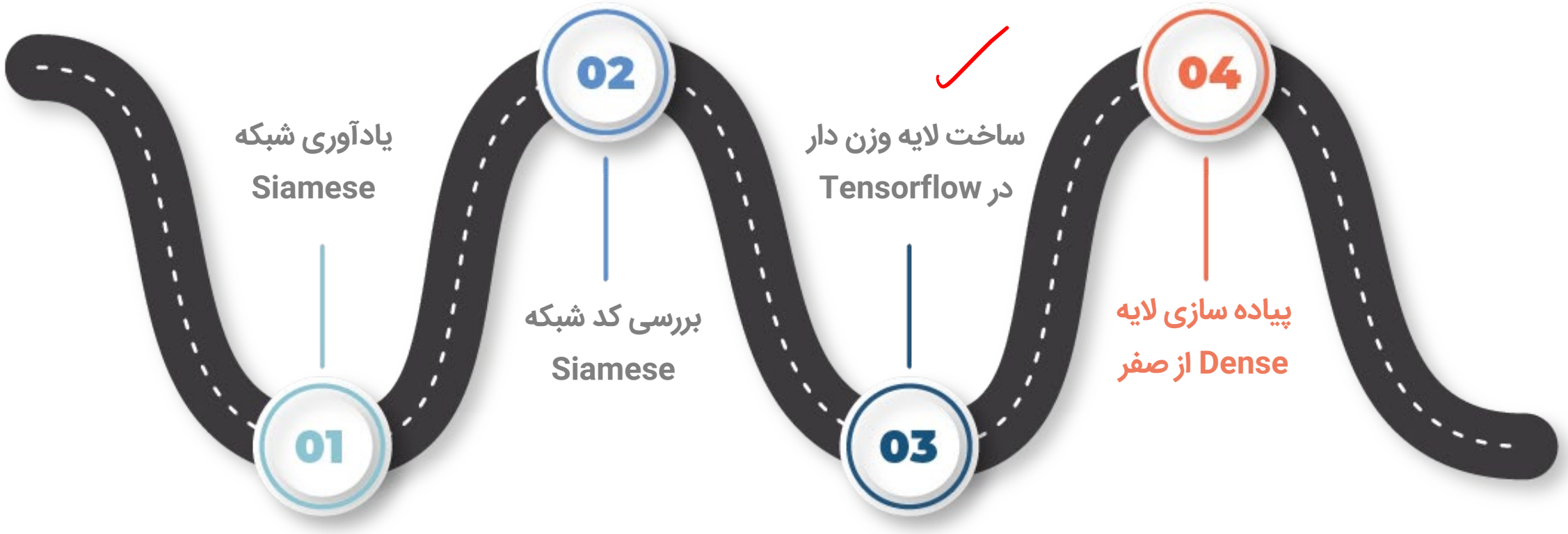
مثلا لایه Dense



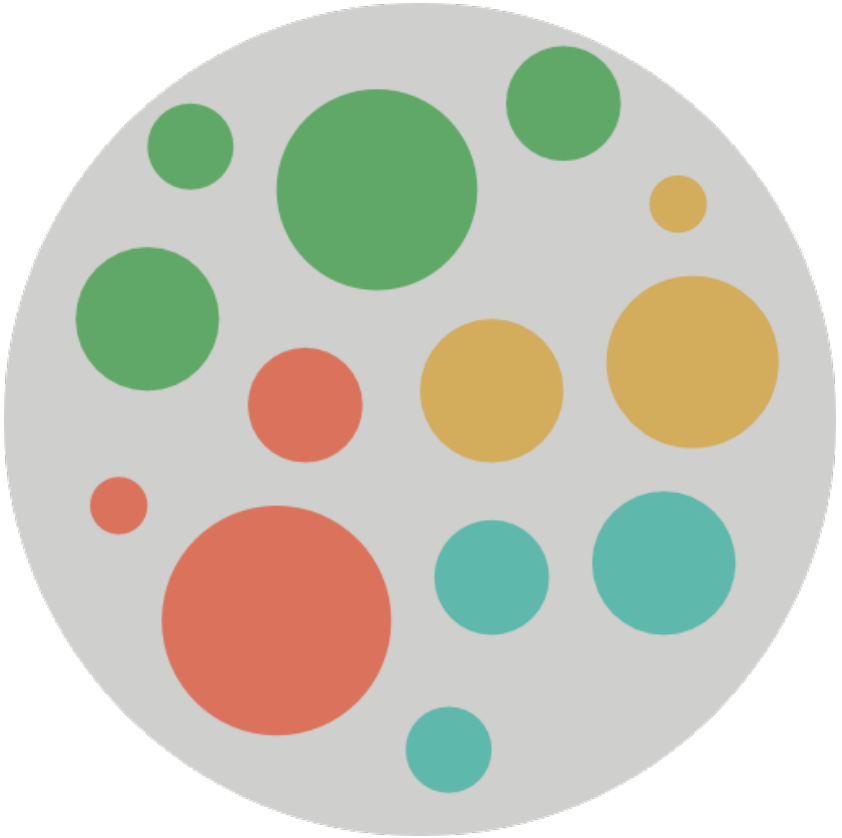
ساختار کلی کد:

```
class NameOfLayer(Layer):

    def __init__(self, ...):
        pass

    def build(self, input_shape):
        pass

    def call(self, inputs):
        pass
```

آنچه امروز خواهیم گفت :



مثال : ساخت لایه Dense

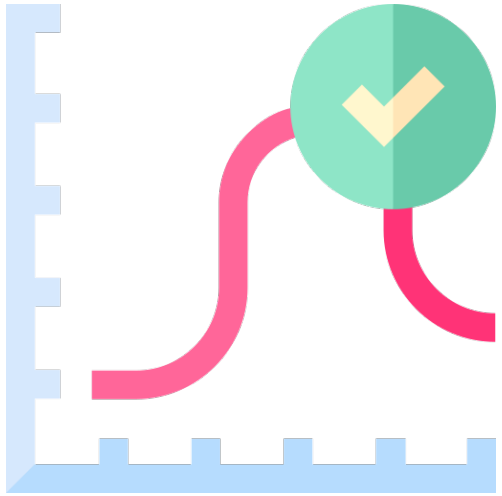


متد __init__

```
def __init__(self, units = 32):
    super().__init__()
    self.units = units
```

tf.random_normal_initalizer

```
tf.random_normal_initializer(
    mean=0.0, stddev=0.05, seed=None
)
```



tf.zeros_initalizer

```
tf.zeros_initializer()
```

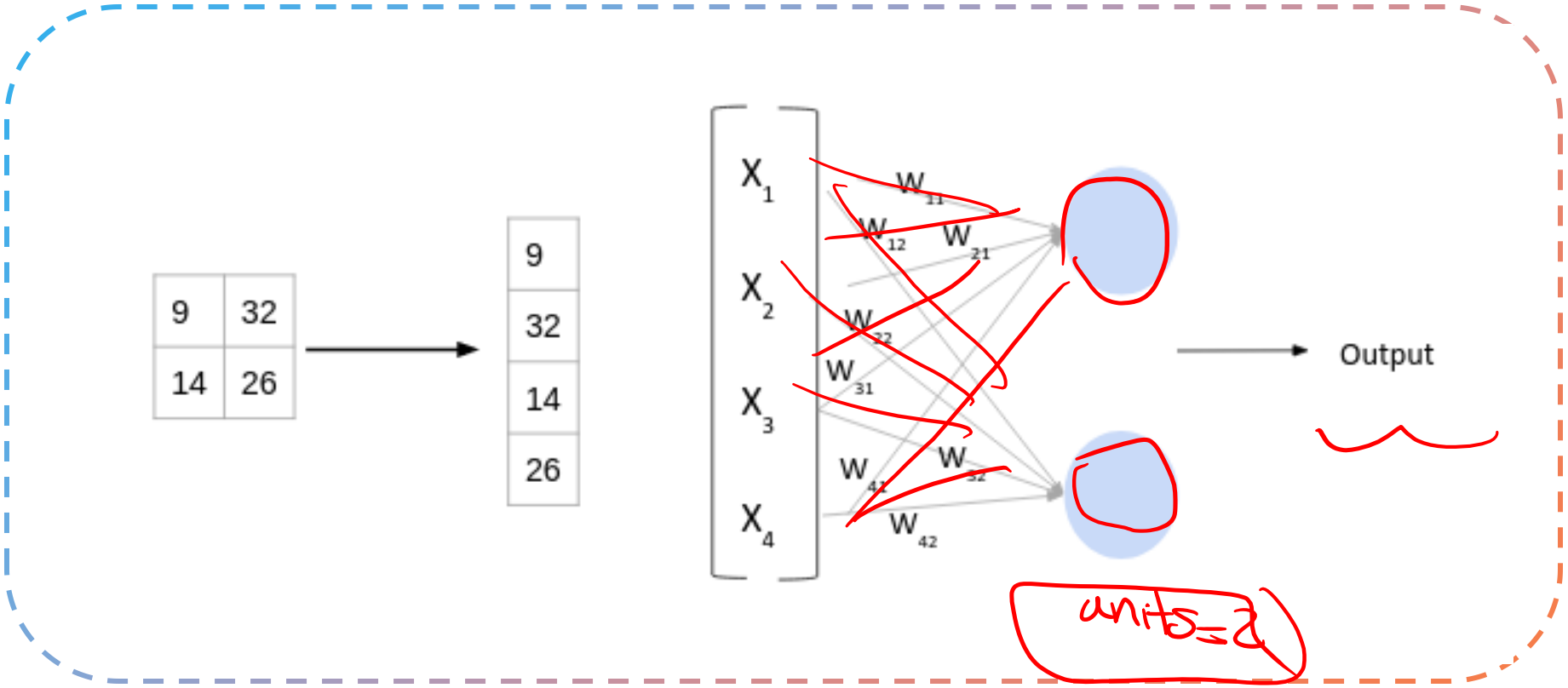
tf.Variable

```
self.w = tf.Variable(initial_value= ...,
                     trainable=...,
                     name = ...)
```

مثال tf.Variable

```
self.w = tf.Variable(initial_value=w_init(shape = (input_shape[-1], self.units),
                     dtype = "float32") ,trainable = True, name="kernel")
```

مروری بر ابعاد در لایه Dense



ابعاد ورودی و خروجی و وزن و بایاس به چه شکل است ؟

مروری بر ابعاد در لایه Dense

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \qquad W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

ابعاد محاسبات در Dense

$$y = x_{(1\times n)} W_{(n\times m)} + b_{(1\times m)}$$

$1\times m$

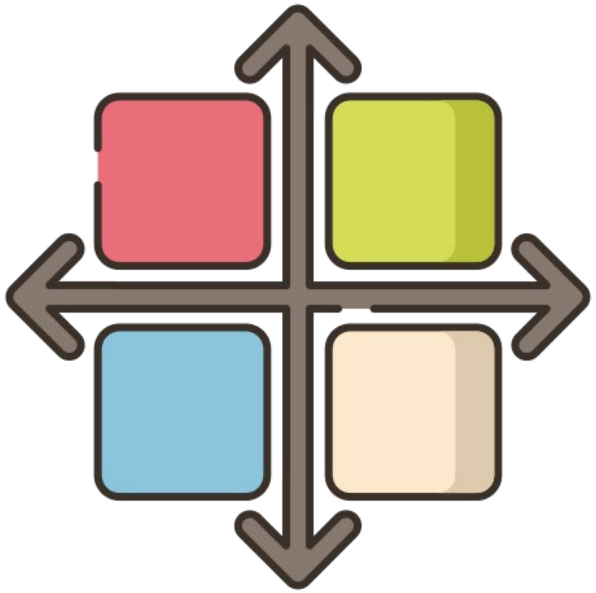
$n : Input\ dim \qquad m : units$

متد __build__

```
def build(self, input_shape):
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(initial_value=w_init(shape = (input_shape[-1], self.units),
                                              dtype = "float32") ,trainable = True, name = "kernel")

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(initial_value=w_init(shape=(self.units,),
                                              dtype="float32"),  trainable =True, name="bias")
```

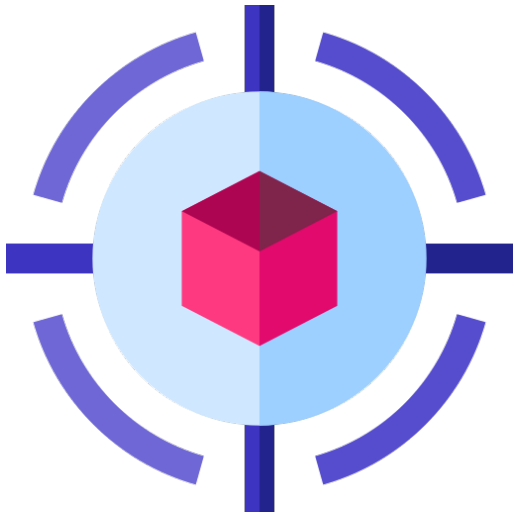
متد call

```
def call(self, inputs):
    return tf.matmul(inputs, self.w) + self.b
```



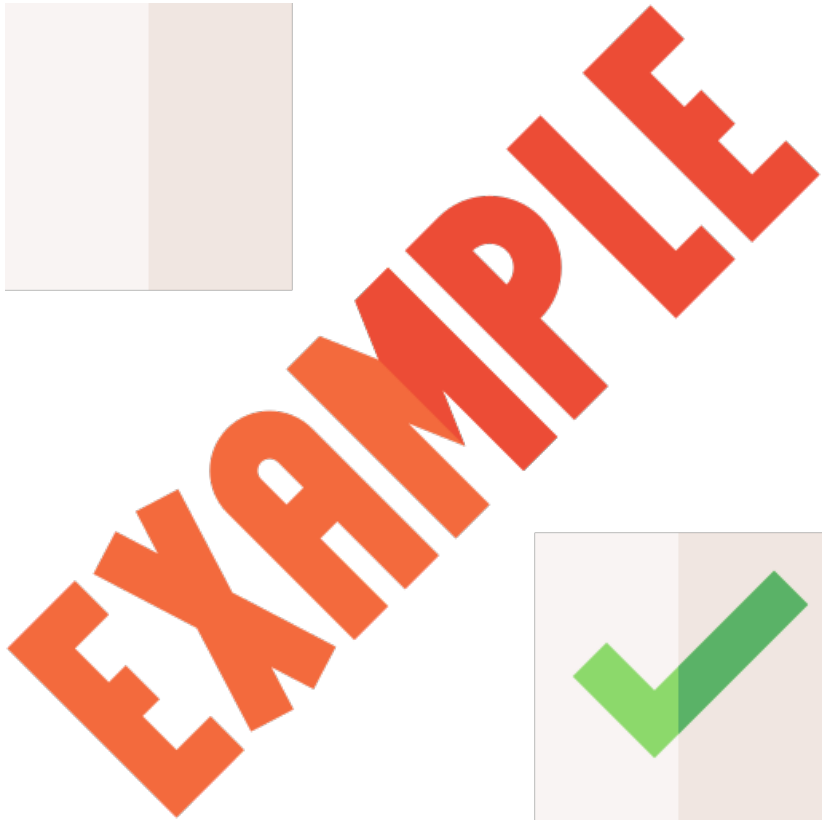
یک بررسی کوچک برای درک بهتر

```
my_dense = SimpleDense(units=1)
x = tf.ones((1,1))
y = my_dense(x)
print(my_dense.variables)
```



$$y = x_{(1\times n)}W_{(n\times m)} + b_{(1\times m)}$$

حل یک مثال:

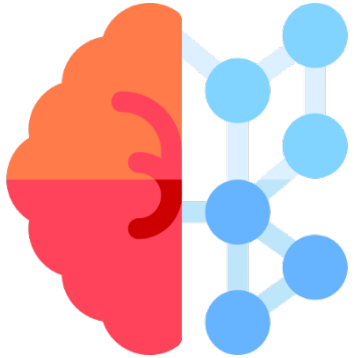


یک مثال ساده:

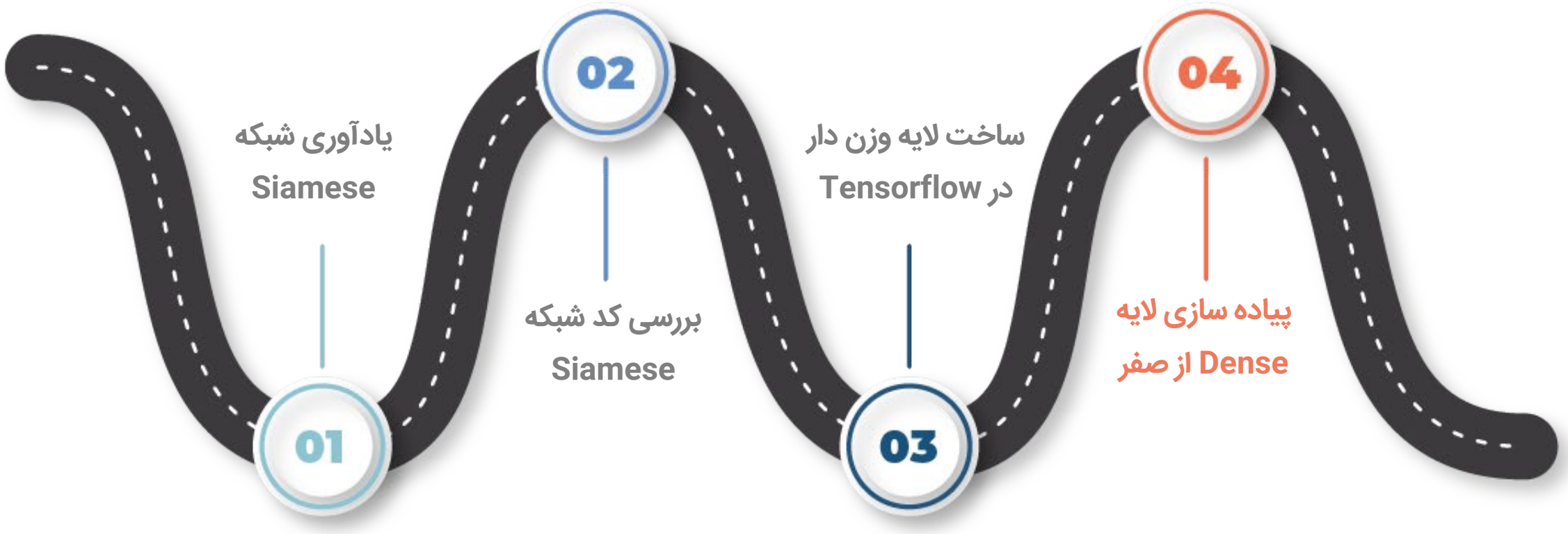
```
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype = float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model = tf.keras.Sequential([
                              SimpleDense(1)
                              ])

model.compile(optimizer="sgd", loss = "mean_squared_error")
model.fit(xs, ys, epochs = 500)
print(model.predict([10.0]))
```

حل یک مثال:

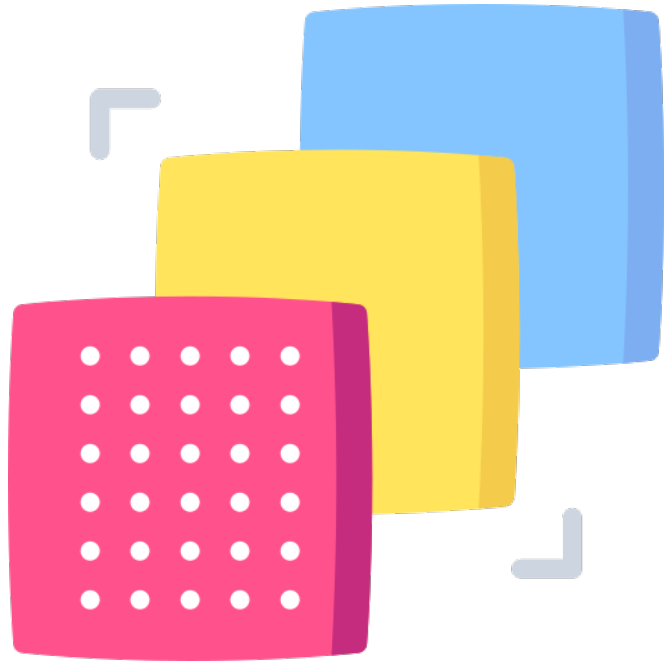
```
net = models.Sequential([
                        layers.Flatten(input_shape= (28 ,28)),
                        SimpleDense(units = 128),
                        layers.Lambda(lambda x:tf.maximum(x, 0.0)),
                        layers.Dropout(0.2),
                        layers.Dense(10, activation="softmax")
                        ])
```



آنچه امروز خواهیم گفت :



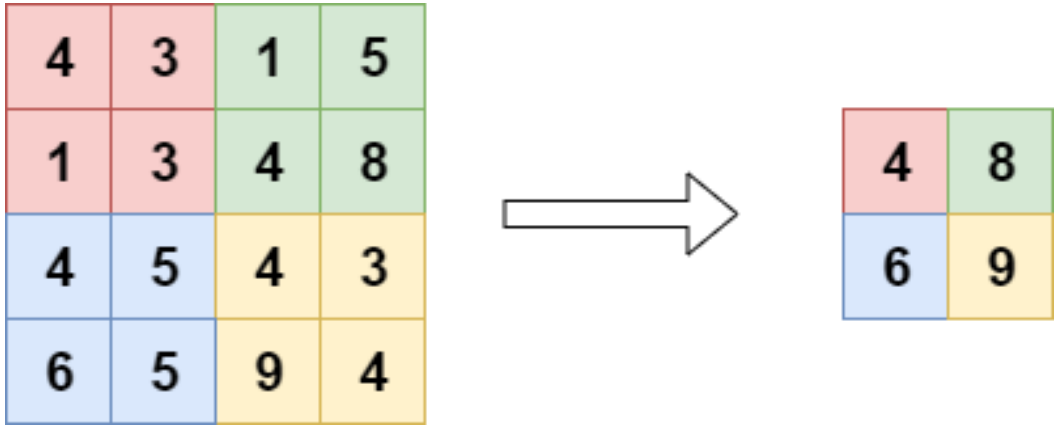
مروری کوتاه بر لایه های pooling



Max Pooling

4	3	1	5
1	3	4	8
4	5	4	3
6	5	9	4

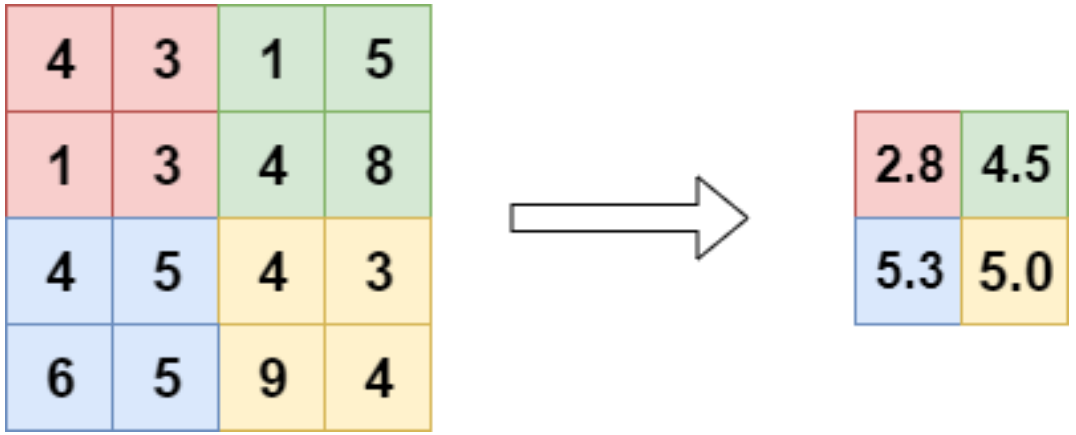
Max([4, 3, 1, 3]) = 4



Average Pooling

4	3	1	5
1	3	4	8
4	5	4	3
6	5	9	4

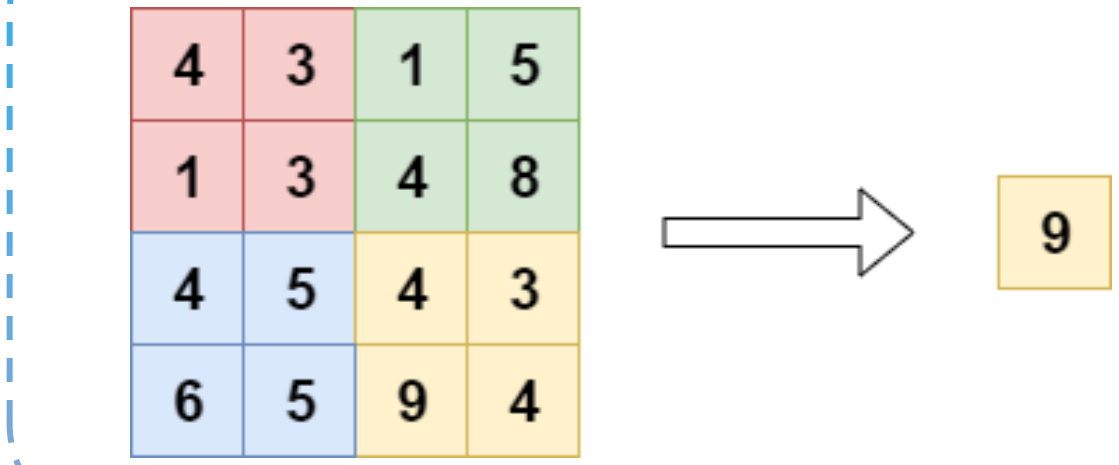
Avg([4, 3, 1, 3]) = 2.75



Global Max Pooling

4	3	1	5
1	3	4	8
4	5	4	3
6	5	9	4

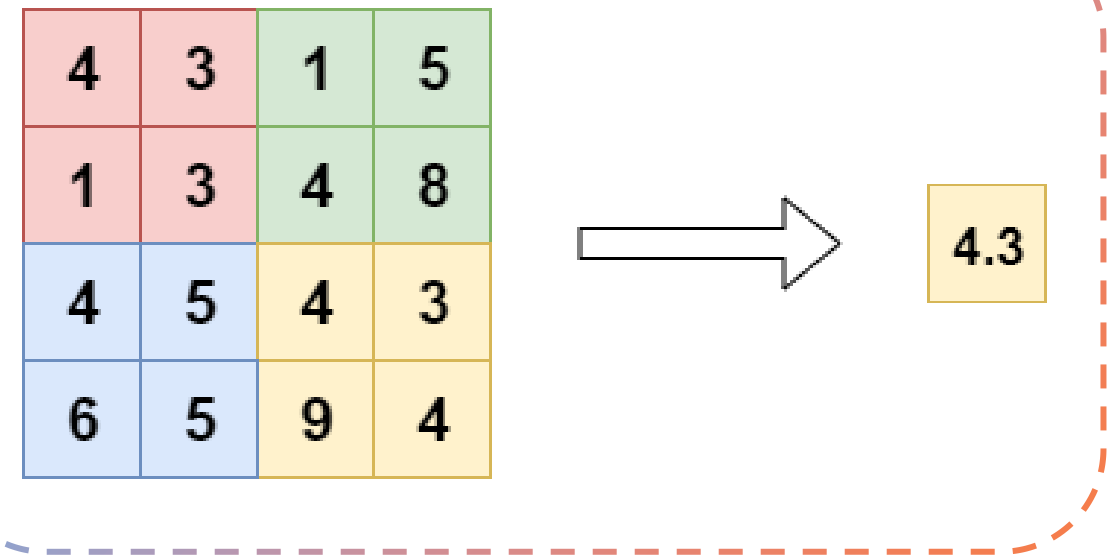
Max([4, 3, 1, 5]
[1, 3, 4, 8]
[4, 5, 4, 3]
[6, 5, 9, 4]) = 9



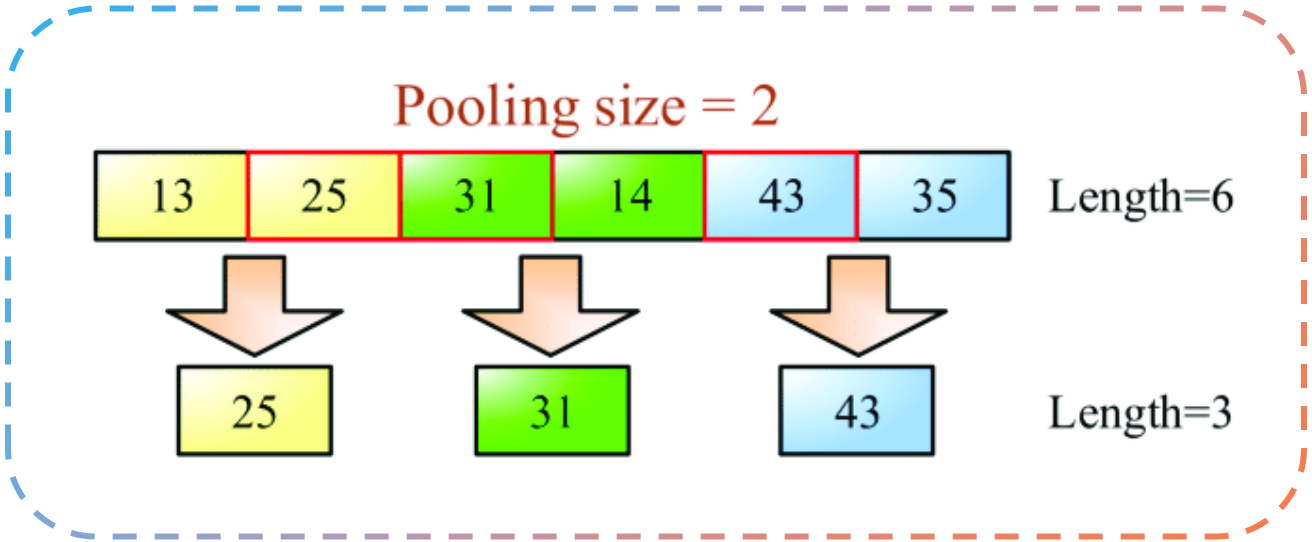
Global Average Pooling

4	3	1	5
1	3	4	8
4	5	4	3
6	5	9	4

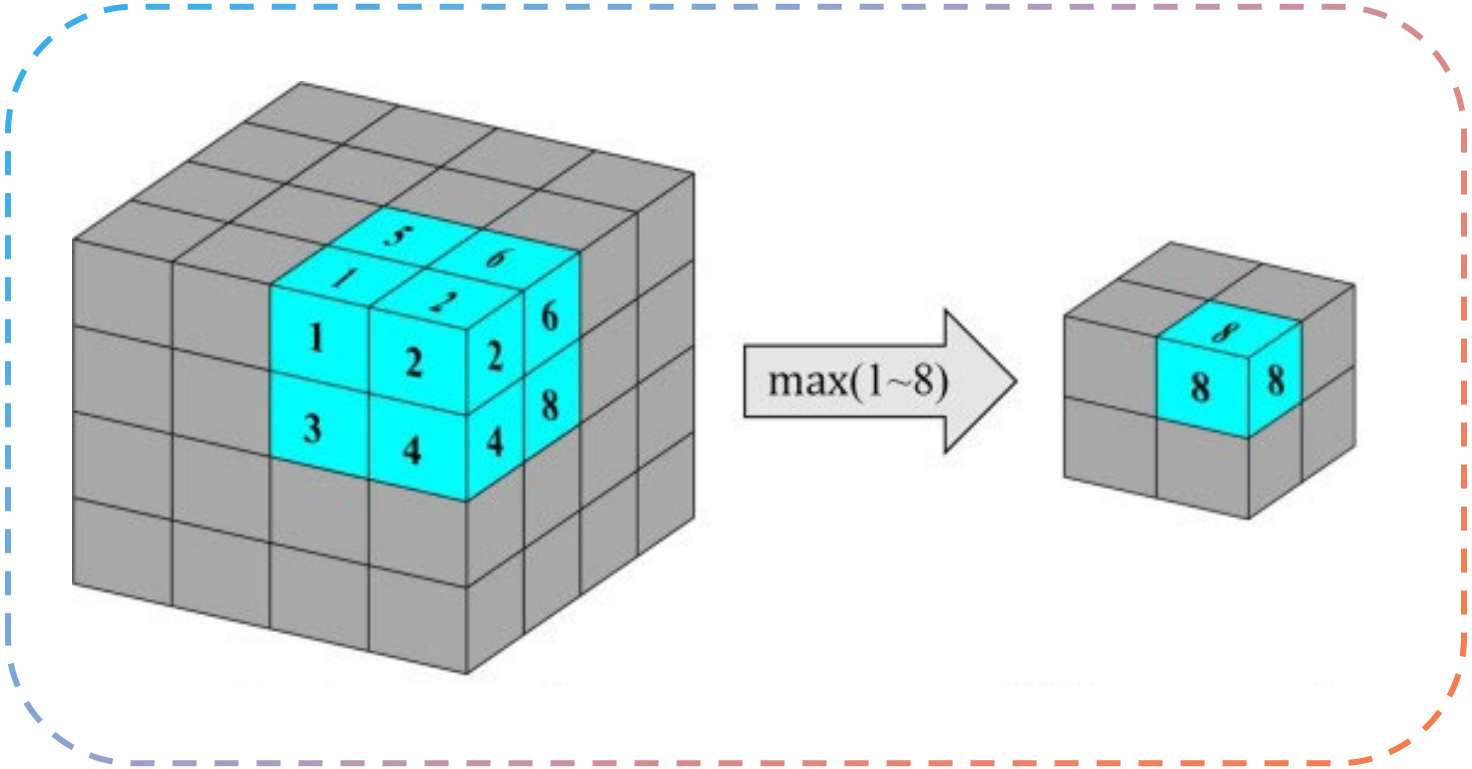
Avg([4, 3, 1, 5] [1, 3, 4, 8] [4, 5, 4, 3] [6, 5, 9, 4]) = 4.3125



الزامی به دو بعدی بودن نیست:



و می تواند سه بعدی هم باشد:



توابع مربوط به pooling در Keras

```
keras.layers.MaxPooling1D(pool_size=2)
keras.layers.MaxPooling2D(pool_size=(2, 2))
keras.layers.MaxPooling3D(pool_size=(2, 2, 2))
```

توابع مربوط به pooling در Keras

```
keras.layers.AveragePooling1D(pool_size=2)
keras.layers.AveragePooling2D(pool_size=(2, 2))
keras.layers.AveragePooling3D(pool_size=(2, 2, 2))
```

توابع مربوط به pooling در Keras

```
keras.layers.GlobalMaxPooling1D(data_format='channels_last')
keras.layers.GlobalMaxPooling2D(data_format='channels_last')
keras.layers.GlobalMaxPooling3D(data_format='channels_last')
```

توابع مربوط به pooling در Keras

```
keras.layers.GlobalAveragePooling1D(data_format='channels_last')
keras.layers.GlobalAveragePooling2D(data_format='channels_last')
keras.layers.GlobalAveragePooling3D(data_format='channels_last')
```